



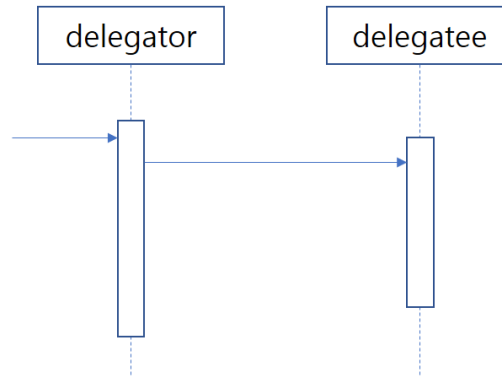
19장. 코틀린의 다양한 기법



19.1. 델리게이션

19.1.1. 델리게이션 클래스

위임 패턴



```
class MyDelegatee {
    fun print(str: String){
        println("i am delegatee : $str")
    }
}
class MyDelegator {
    val delegatee: MyDelegatee = MyDelegatee()

    fun print(str: String){
        delegatee.print(str)
    }
}
fun main(args: Array<String>) {
    val obj: MyDelegator = MyDelegator()
    obj.print("hello kang")
}
```

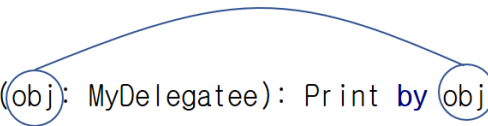
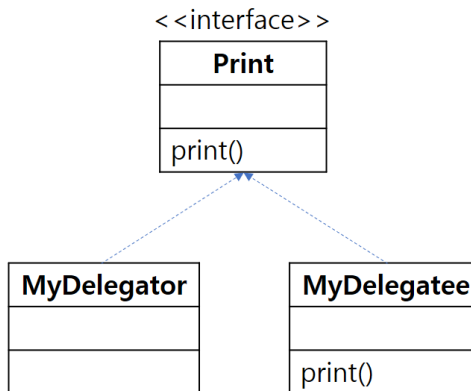
19.1. 델리게이션

코틀린에서는 위임을 by 예약어를 이용

```
interface Print {  
    fun print(arg: String)  
}  
  
class MyDelegatee: Print {  
    override fun print(arg: String) {  
        println("i am delegatee : $arg")  
    }  
}  
  
class MyDelegator(obj: MyDelegatee): Print by obj  
  
fun main(args: Array<String>) {  
    val obj: MyDelegatee = MyDelegatee()  
    MyDelegator(obj).print("hello kkang")  
}
```

19.1. 델리게이션

```
class MyDelegator (obj: MyDelegatee): Print by obj
```

A diagram showing a curved line connecting the `obj` parameter in the class definition to the `by obj` part of the `Print` trait, indicating that the trait is delegated to the object.

19.1. 델리게이션

19.1.2. 델리게이션 프로퍼티

- 프로퍼티의 위임도 `by`에 의해 제공
- 프로퍼티의 위임을 위한 클래스는 `getValue()` 와 `setValue()` 함수를 가지고 있어야 한다.
- 프로퍼티의 `get()` 에 의해 `getValue()` 함수가, `set()` 에 의해 `setValue()` 함수가 호출

```
class MySumDelegate {
    var result: Int = 0
    operator fun getValue(thisRef: Any?, property: KProperty<*>): Int {
        println("getValue call... ref : $thisRef, property : '${property.name}")
        return result;
    }
    operator fun setValue(thisRef: Any?, property: KProperty<*>, value: Int) {
        result=0
        println("setValue call... value : $value, '${property.name}")
        for(i in 1..value){
            result += i
        }
    }
}

class Test {
    var sum: Int by MySumDelegate()
}
```

19.2. SAM 전환

19.2.1. 자바에서 인터페이스 활용

- SAM(Single Abstract Method) 는 자바 API를 코틀린에서 활용할 때 람다 표현식을 이용해 쉽게 이용할수 있게 제공하는 기법
- SAM은 단어 뜻 그대로 하나의 추상함수를 가지는 인터페이스의 활용을 목적
- 인터페이스가 자바에 작성
- 인터페이스를 등록하는 setter 함수가 자바에 작성

19.2. SAM 전환

```
public interface JavaInterface1 {  
    void callback();  
}
```

```
public class SAMTest {  
    JavaInterface1 callback;  
    public void setInterface(JavaInterface1 callback){  
        this.callback=callback;  
    }  
}
```

```
SAMTest obj=new SAMTest();  
obj.setInterface(new JavaInterface1() {  
    @Override  
    public void callback() {  
        System.out.println("hello java");  
    }  
});
```


19.2. SAM 전환

19.2.2. 코틀린에서 SAM 활용

```
fun main(args: Array<String>) {  
    val obj=SAMTest()  
    obj.setInterface(object : JavaInterface1{  
        override fun callback() {  
            println("hello kotlin")  
        }  
    })  
    obj.callback.callback()  
}
```

```
fun main(args: Array<String>) {  
    val obj=SAMTest()  
  
    obj.setInterface { println("hello sam") }  
    obj.callback.callback()  
}
```

obj.setInterface { println("hello sam") }



obj.setInterface(
 object : JavaInterface1{
 override fun callback(){
 println("hello sam")
 }
 }
)

컴파일러에 의해 자동 추가

19.3. 타입 에일리어스

- 타입에 대한 이름 변경

```
typealias MyInt = Int
typealias MList<T> = MutableList<T>
typealias MC = MyClass
typealias MI = MyInterface

interface MyInterface
class MyClass: MI

fun main(args: Array<String>) {
    val no: MyInt = 10
    val list: MList<String> = mutableListOf("hello", "kang")
    val obj: MC = MC()
}
```

- 함수타입

```
typealias MyType = (Int) -> Boolean
val myFun: MyType = { it > 10 }
```

19.3. 타입 에일리어스

- 이너클래스

```
class Super {  
    inner class Sub  
  
    fun getSubInstance(): MySub {  
        return Sub()  
    }  
}  
  
typealias MySub = Super.Sub
```